NOV 2020

PROJECT REPORT

FOR

E-MAIL SPAM CLASSIFIER

PREPARED BY

MAYANK JHANWAR
RegNo. : 189303179
Email : mayank.189303179@muj.manipal.edu

NAMIT VARSHNEY
RegNo. : 189303111
Email : namit.189303111@muj.manipal.edu

MANIPAL UNIVERSITY
JAIPUR

Established under the Manipal University Jaipur Act (No. 21 of 2011)

INDEX

MANIPAL UNIVERSITY
JAIPUR

Established under the Manipal University Jaipur Act (No. 21 of 2011)

CERTIFICATE

This is to certify that the project entitled

"**E-Mail Spam Classifier**"

is a bonafide work carried out as part of the course

Data Analysis and Algorithm (DAA)

under my guidance by

Mayank Jhanwar
Reg. No: 189303179

Namit Varshney
Reg. No: 189303111

student of **B.Tech - V Semester** at the
Department of Computer and Communication Engineering , Manipal University Jaipur,

during the academic **Semester V** , in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer and Communication Engineering**, at
Manipal University Jaipur

Month of Submission
Nov 2020

Signature of Instructor
Dr. Geeta Rani

1 | Introduction

1.1 Purpose

Spamming is one of the major attacks that accumulate many compromised machines by sending unwanted messages, viruses, and phishing through E-Mails**.** It conveys the principal aim behind choosing this project under with. There are many people who try to fool you, just by sending you fake e-mails! Some of them have the messages like - You have won 1000 dollars, or we have deposited this much amount to your account. Once you open this link, then they will track you and try to hack your information.

The scope for this project is to identify all spam e-mails with the help of Machine Learning Algorithms.

1.2 Document Conventions

There are no used conventions till now.

1.3 Intended Audience and Reading Suggestions

We intend this document for anyone (such as developers, project managers, programmers, users, testers) who is interested in knowing about an E-Mail Spam Classifier.
The SRS contain answers for the following questions:

- **WHY an E-mail Spam Classifier is made?**
- **HOW an E-mail Spam Classifier is made?**
- **WHAT are the requirements for an E-mail Spam Classifier?**

1.4 Product Scope

Spamming is a big problem! Anyone who is connected to internet faces this problem now and then, but with Corporates, it becomes a serious problem. Phishing attacks, greedy traps and many create a havoc if not handled properly.
Our product **E-Mail Spam Classifier** is a solution for these Spams. Through our Intelligent Web- Based Product, you can find Spam E-Mails and can label them as Spam for future reference.

1.5 References

1.5.1 Websites

- www.kaggle.com
- www.sci-hub.tw
- www.ieeexplore.ieee.org
- www.medium.com

1.5.2 Dataset

- https://www.kaggle.com/venky73/spam-mails-dataset

2 | Overall Description

2.1 Product Perspective

The context of this product is as described above, and our team's goal is to make a safer and crime-free Cyber-World.
This is the first time we are working on this product, and it is something new for us. We are fully aware of the fact that there will be several challenges to make this product efficient and trustworthy. As spammers are getting smarter each day, we also need highly efficient Machine Learning Algorithm to tackle and knock out these Spammers.

2.2 Product Functions

Major highlights of our product's functionality are:

- This product will help **identify Spam E-Mails** similar to the Spam encountered earlier, which are stored in a vast library of Spam E-Mails.
- This product will also help in **identifying new Potential Spam E-Mails** from known & unknown sources. This what is going to be a speciality of our product.

2.3 User Classes and Characteristics

Our user class is vastly distributed across all kinds of Computer-related Industries. Some of them are mentioned below:

- **General Public**:
  Our Sole important target is common people who use E-Mail daily for their works and general internet communications. These are the one who gets the most E-Mail spams, and according to surveys, the public is the most favourite targets of these Spammers.
- **Corporate Companies**:
  Next big user class for this Product will be Corporate Companies which, because of employee's negligence and inadequate knowledge about Spam E-mail and trap their traps, makes companies' data vulnerable and pose great economic losses.

3 | External Interface Requirements

3.1 User Interface

The User Interface of our Project is simple, user-friendly, and self-explanatory. As per our current product Idea, we have a basic Website which is being hosted on the internet globally which has certain key options to check Spam E-Mails.

3.2 Hardware Interfaces

The Hardware required to run this Product is very minimal and is mentioned below:

- **Operating System – Win/Mac/Linux (any distribution)**
- **Web Browser–Chrome/Mozilla/Opera/Tor (Updated with latest FTP/HTTP support)**
- **Proper Internet Connection (above 1 Mbps)**

3.3 Software Interfaces

There are tons of software which are available online which can be used to make a Spam Classifier. But we have mentioned some softwares which we have used and will use in our Development of this Product, which are mentioned below:

- **Jupyter Notebook | Google Collab**
- **PyCharm**
- **Web Browser (Chrome)**

3.4 Technologies Used:

There are several technologies on which we worked, and these are:

- **For ML Classification**
  1) NumPy
  2) Pandas
  3) NLTK
  4) Sckit-Learn

Building ML Model

Steps with Explanation

Step 1: All Required libraries should be imported.

```
#Import libraries
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
import string
```

The usage of the libraries imported can be explained as:

- **Numpy** - for handling data arrays and making matrices.
- **Pandas** - for analyzing and manipulating data.
- **Natural Language Toolkit(NLTK)** - for accessing text processing libraries like tokenization.
- **Stopwords** - removes useless words(data).
- **Strings** - for accessing string operations.

Step 2: Uploading the data in the colab notebook and printing the first five rows.

```
df = pd.read_csv('spam_ham_dataset.csv')
df.head(5)
```

	Unnamed: 0	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	3624	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	4685	spam	Subject: photoshop , windows , office . cheap ...	1
4	2030	ham	Subject: re : indian springs\r\nthis deal is t...	0

Step 3: Exploring the data and getting the number of rows & columns and columns names.

```
#Print the shape (Get the number of rows and cols)
print(df.shape)
#Get the column names
df.columns
```

```
(5171, 4)
Index(['Unnamed: 0', 'label', 'text', 'label_num'], dtype='object')
```

Step 4: Checking for duplicates and removing them.

```
#Checking for duplicates and removing them
df.drop_duplicates(inplace = True)
#Show the new shape (number of rows & columns)
df.shape
```

```
(5171, 4)
```

Note: Since there were no duplicates, the shape of the dataset remains the same.

Building ML Model

Steps with Explanation - Continued...

Step 5: Printing the number of missing data for each column.

```
#Show the number of missing (NAN, NaN, na) data for each column
print(df.isnull().sum())
#Need to download stopwords
nltk.download('stopwords')
```

```
Unnamed: 0    0
label         0
text          0
label_num     0
dtype: int64
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Note: Since there was no missing data, all the entries are zero. Downloading the stop words. Stop words in natural language processing are useless words (data).

Step 6: Creating a function to clean the text and return the tokens.

```
#Tokenization (a list of tokens), will be used as the analyzer
#1.Punctuations are [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~]
#2.Stop words in natural language processing, are useless words (data).
def process_text(text):

    #1 Remove Punctuationa
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)

    #2 Remove Stop Words
    clean_words = [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]

    #3 Return a list of clean words
    return clean_words
```

Note: The cleaning of the text can be done by first removing punctuations and then removing the useless words also known as stop words.

Step 7: Show the Tokenization (a list of tokens)

```
#Show the Tokenization (a list of tokens )
print(df['text'].head().apply(process_text))
from sklearn.feature_extraction.text import CountVectorizer
messages_bow = CountVectorizer(analyzer=process_text).fit_transform(df['text'])
```

```
0    [Subject, enron, methanol, meter, 988291, foll...
1    [Subject, hpl, nom, january, 9, 2001, see, att...
2    [Subject, neon, retreat, ho, ho, ho, around, w...
3    [Subject, photoshop, windows, office, cheap, m...
4    [Subject, indian, springs, deal, book, teco, p...
Name: text, dtype: object
```

Note: The process of returning tokens from text is known as Tokenization. Printing the Tokenization of the first 5 rows of text data from our data set by applying the function process_text and Converting the text into a matrix of token counts.

Building ML Model

Steps with Explanation - Continued...

Step 8: Split the data into training & testing sets, and print them

```
#Split data into 80% training & 20% testing data sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(messages_bow, df['label'], test_size = 0.20, random_state = 0)
#Get the shape of messages_bow
messages_bow.shape
```

```
(5171, 50381)
```

Note: We will use this one row of data for testing to make our prediction later on and test to see if the prediction matches with the actual value. The testing feature (independent) data set will be stored in x_test and the testing target (dependent) data set will be stored in y_test . The training feature (independent) data set will be stored in x_train and the training target (dependent) data set will be stored in y_train. Printing the shape of the modified dataset.

Step 9: Creating and training the Multinomial Naive Bayes classifier

```
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Note: Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification).

Step 10: Checking the classifiers prediction and actual values on the data set.

```
#Print the predictions
print(classifier.predict(X_train))
#Print the actual values
print(y_train.values)
```

```
['ham' 'ham' 'ham' ... 'spam' 'ham' 'ham']
['ham' 'ham' 'ham' ... 'spam' 'ham' 'ham']
```

Step 11: Showing the report, confusion matrix & accuracy score.

```
#Evaluate the model on the test data set
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
pred = classifier.predict(X_test)
print(classification_report(y_test ,pred ))
print('Confusion Matrix: \n', confusion_matrix(y_test,pred))
print()
print('Accuracy: ', accuracy_score(y_test,pred))
```

```
              precision    recall  f1-score   support

         ham       0.98      0.98      0.98       732
        spam       0.95      0.96      0.96       303

    accuracy                           0.97      1035
   macro avg       0.97      0.97      0.97      1035
weighted avg       0.97      0.97      0.97      1035
```

Building ML Model

Steps with Explanation - Continued...

Step 12: Testing the model / classifier

```
#Print the predictions
print('Predicted value: ',classifier.predict(X_test))
#Print Actual Label
print('Actual value: ',y_test.values)
```

```
Predicted value:  ['ham' 'ham' 'ham' ... 'ham' 'spam' 'ham']
Actual value:  ['ham' 'ham' 'ham' ... 'ham' 'spam' 'ham']
```

Note: It looks like the model / classifier used is 98.74% accurate. Testing the model / classifier on the test data set(x_train and y_train) by printing the predicted value, and the actual value to see if the model can accurately classify the email text/message.

Step 13: Evaluating the model on the test data set.

```
#Evaluate the model on the test data set
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
pred = classifier.predict(X_test)
print(classification_report(y_test ,pred ))
print('Confusion Matrix: \n', confusion_matrix(y_test,pred))
print()
print('Accuracy: ', accuracy_score(y_test,pred))
```

```
              precision    recall  f1-score   support

         ham       0.98      0.98      0.98       732
        spam       0.95      0.96      0.96       303

    accuracy                           0.97      1035
   macro avg       0.97      0.97      0.97      1035
weighted avg       0.97      0.97      0.97      1035

Confusion Matrix:
 [[718  14]
 [ 13 290]]

Accuracy:  0.9739130434782609
```

Note: The classifier accurately identified the email messages as spam or not spam with 97.39 % accuracy on the test data.

Step 14: Saving the ML model for importing into Flask.

```
import joblib
filename = 'finalized_model.sav'
joblib.dump(classifier, filename)
```

```
['finalized_model.sav']
```

Making Project Functional

Linking Machine Learning Model with Frontend

This section discusses in detail, about how we made our Project Functional with Graphical User Interface :

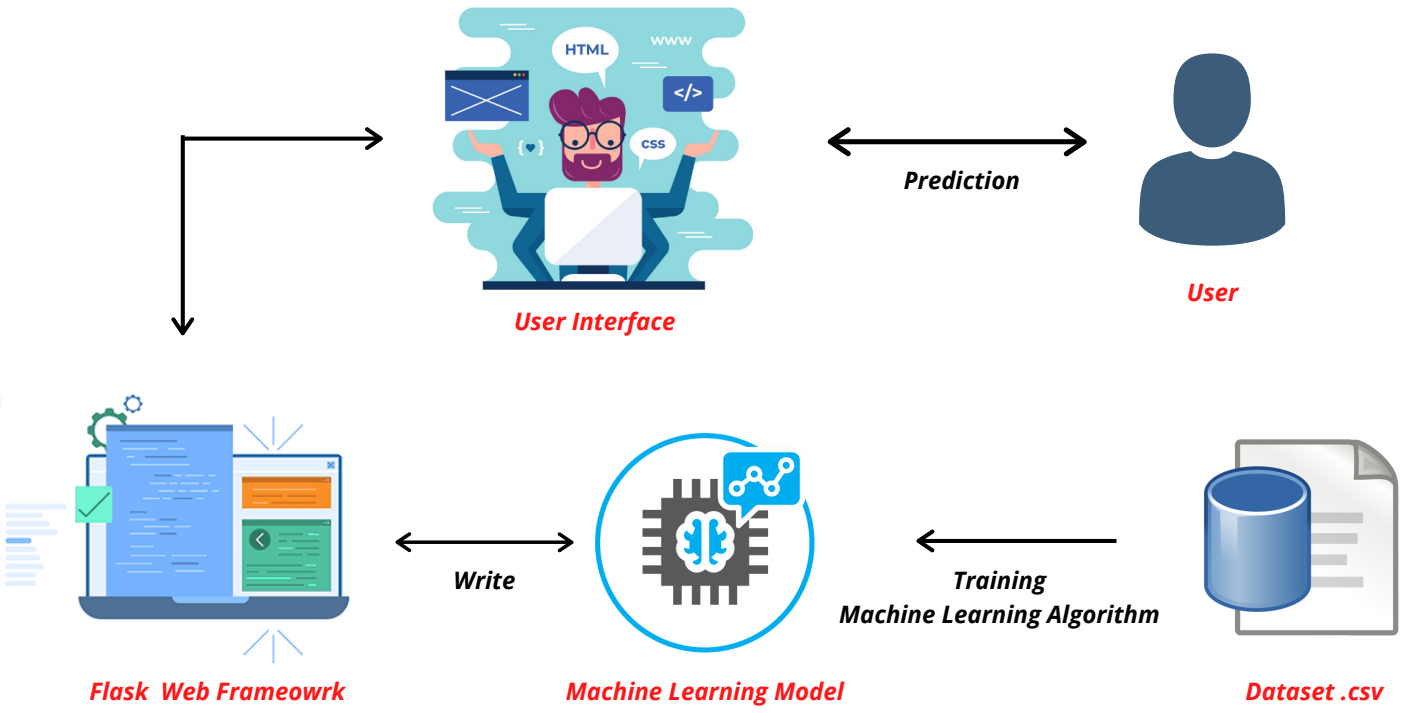
Front-end:
The project's GUI has been made through web pages written in HTML and design elegantly with CSS and JavaScript for it to be attractive.

Back-end Framework:
Flask helps in implementing a machine learning application in Python that can be easily plugged, extended and deployed as a web application.

Machine Learning Model:
The model detects, if a text message/mail is spam(1) or ham(0) using techniques like tokenizaton, multinomial naive bayes classifier, etc.

Web Server*:*
All these functional units are connected by establishing a server on the web by hosting it on Python Everywhere, which helps us in running the project successfully..

The linkage can be better understood by referring the diagram shown below:



Hosting our Python ML Model

on Web through Flask Framework

As our machine learning model is predicting accurate results, We are ready to expose it to the outside world. This section explains the steps which we followed to host a machine learning model to the outside world. Public can now access our work via their web browsers. It is an important step for anyone who wants to make a business out of their machine learning models.

Flask Framework

Flask acts as a collection of software packages which can help us easily create a web application. It helps in implementing a machine learning application in Python that can be easily plugged, extended and deployed as a web application. Flask is based on two key components: WSGI toolkit and Jinja2 template engine. WSGI is a specification for web applications and Jinja2 renders web pages.

Step 1: Building a Machine Learning Model

As we have already implemented a machine learning model in the previous section that takes in an input and outputs a variable. For a quick recap, **our machine learning model takes in an input i.e. any mail received and gives us the predicted output as "Spam" or "Not Spam".**

Hence our machine learning model looks like:

```
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
output = classifier.predict(X_train)  #output
```

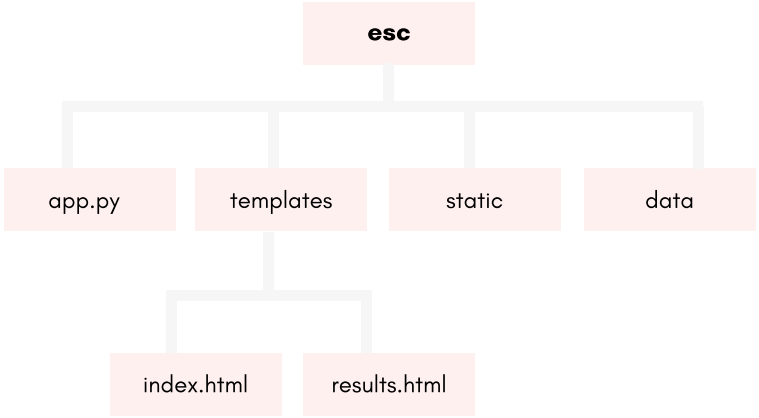
Step 2: Installing Flask

Installing Flask is easy and can be done by just opening the command prompt and running the following command:

```
pip install flask
```

Step 3: Creating a Folder structure

Creating a folder structure is important. We will have to create 4 sub sections as shown below:



app.py:
This Python file will contain the Flask specific code.
templates:
This folder will contain the HTML files. These HTML files will be rendered on the web browser.
static:
This folder will contain the CSS, JS files. These files will also be rendered on the web browser.
data:
This folder will contain the saved Machine Learning model and dataset.

Embedding Flask into app.py

Creating a file app.py and perform the following steps:

Step 1: Importing libraries

```
from flask import Flask,render_template,url_for,request
import pandas as pd
import pickle
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
import joblib
```

Step 2: Instantiate Flask

```
app = Flask (__name__)
```

Step 3: Setting up Routes

We are going to use Python decorators to decorate a function so that the URL is mapped to a function:

```
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict',methods=['POST'])
def predict():
    df= pd.read_csv("spam_ham_dataset.csv")
    df_data = df[["text","label_num"]]

    # Features and Labels
    df_x = df_data['text']
    df_y = df_data.label_num

    # Extract Feature With CountVectorizer
    corpus = df_x
    cv = CountVectorizer()
    X = cv.fit_transform(corpus) # Fit the Data
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, df_y, test_size=0.33, random_state=42)

    #Naive Bayes Classifier
    from sklearn.naive_bayes import MultinomialNB
    clf = MultinomialNB()
    clf.fit(X_train,y_train)
    clf.score(X_test,y_test)

    #Alternative Usage of Saved Model
    #ytb_model = open("spam_mail_detection.py","rb")
    #clf = joblib.load(ytb_model)
```

Embedding Flask into app.py

Explanation:

In the above Flask code, "/" URL is now mapped to the render_template function. Therefore if a user visits "/", then Flask will render index.html on the web browser of the user. Flask will look for the html file in the templates folder.

Flask is based on Jinja2 template engine. The render_template() function renders the template and expects it to be stored in the Templates folder on the same level as the app.py file.

Next the function is now mapped to predict() function which will load the pre-trained model. Subsequently, it will then get the entered mail by the user and will display the predicted value as "SPAM" or "NOT SPAM" on the web page by rendering the results.html file.

Note how we are passing text and label_num to the render_template() function as shown below:

```
df= pd.read_csv("spam_ham_dataset.csv")
df_data = df[["text","label_num"]]
```

Step 4: Code to display results on Results.html

```
if request.method == 'POST':
    comment = request.form['comment']
    data = [comment]
    vect = cv.transform(data).toarray()
    my_prediction = clf.predict(vect)

return render_template('results.html',prediction = my_prediction)
```

Step 5: Run the Module

```
if __name__ == '__main__':
app.run(debug=True)
```

What happens when app,run() is executed?

1. When the app.run() is executed then the application will start running. It will run on the host "localhost" which is your local machine on the port number 9999.
2. The debug flag is used during development so that the Flask server can reload the code without restarting the application.
3. It also outputs useful debugging information. Therefore, when the user will visit http://localhost:9999/, Flask will render the index.html page.

implementing .html pages

Steps:

Step 6: Implement index.html

```
<div class="get-start-area">
    <!-- Form Start -->
        <form action="{{ url_for('predict')}}" method="POST" class="form-inline">
            <textarea name="comment" class="form-control" id="comment" cols="50" rows="4"
              placeholder="Enter Text Message*" required></textarea>
            <input type="submit" class="submit" value="Check Spam or Not!">
        </form>
    <!-- Form End -->
</div>
```

The key to note is the action attribute. It is set to **"{{ url_for('predict')}}".** So, when a user enters an E-mail and submits it. The POST method stores it in the **name** attribute named "comment" in the above html code & then passes it to the render_template() function.

Step 7: Implement results.html

```
<div class="wellcome-heading">
    {% if prediction == 1%}
        <h2 style="color:white;">Spam</h2>
    {% elif prediction == 0%}
        <h2 style="color:white;">Not a Spam!</h2>
    {% endif %}
</div>
```

The **prediction** containers will display the predicted spam or not spam result. These values will be passed in via the render_template() function in the app.py file.

Step 8: run the application to localhost

Now, go to the project directory in the CMD and type:

```
python app.py
```

This will now run your python application using Flask micro framework on your local machine. Flask will run the application on port 9999 therefore navigate to:

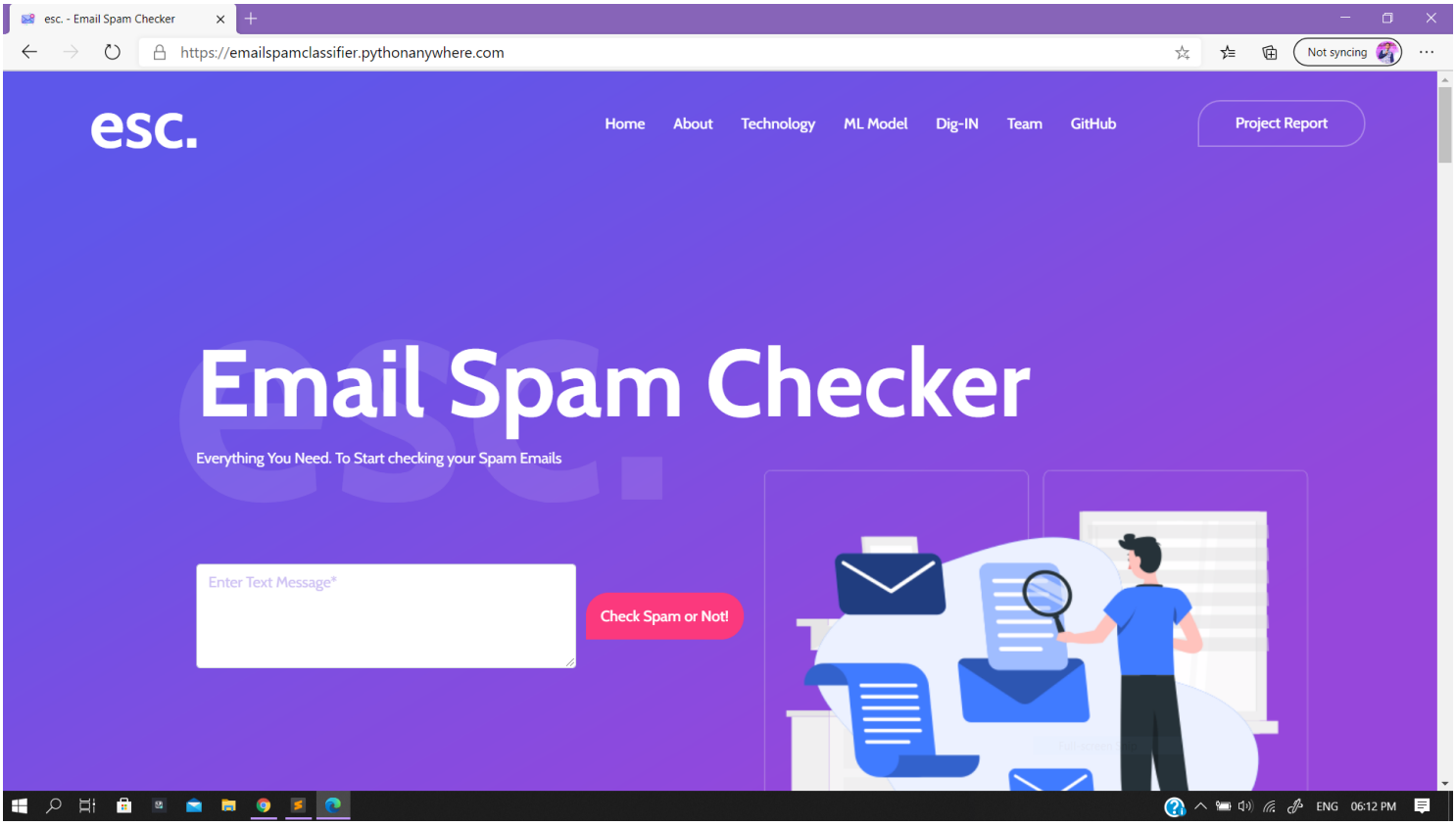
```
http://localhost:9999/
```

Our model is now running on a web browser on our localhost. The last task is to deploy it on an external server so that the public can access it. For this Project we have, hosted our project on **pythonanywhere.com,** by uploading the project folders on the web console in the structure mentioned and by installing all the libraries on the bash console. This project can now be accessed by visiting the following url:

https://emailspamclassifier.pythonanywhere.com/

user interface

index.html



results.html